Secure APIs For Applications In Microkernel-Based Systems

Mohammad Hamad , and Vassilis Prevelakis
Institute of Computer and Network Engineering, Technical University of Braunschweig,Braunschweig, Germany
{*mhamad,prevelakis*}*@ida.ing.tu-bs.de*

Keywords: Security, Secure APIs .

Abstract: The Internet evolved from a collection of computers to today's agglomeration of all sort of devices (e.g. printers, phones, coffee makers, cameras and so on) a large part of which contain security vulnerabilities. The current wide scale attacks are, in most cases, simple replays of the original Morris Worm of the mid-80s. The effects of these attacks are equally devastating because they affect huge numbers of connected devices. The reason for this lack of progress is that software developers will keep writing vulnerable software due to problems associated with the way software is designed and implemented and market realities. So in order to contain the problem we need effective control of network communications and more specifically, we need to vet all network connections made by an application on the premise that if we can prevent an attacker from reaching his victim, the attack cannot take place. This paper presents a comprehensive network security framework, including a well-defined applications programming interface (API) that allows fine-grained and flexible control of network connections. In this way, we can finally instantiate the principles of dynamic network control and protect vulnerable applications from network attacks.

1 INTRODUCTION

Every day we come across examples of security failures that cast doubt on the reliability of today's IT infrastructure. We hear about compromises spanning huge numbers of IoT devices, or about hundreds of thousands of customer records lost. All these failures largely stem from the ability of Internet-aware devices (or hosts) to be contacted by any other host connected to the Internet. Our lack of effective management of these connections is allowing malfeasants to connect to our devices and cause havoc. Another problem lies with the applications themselves. Even when extensively tested, they often contain vulnerabilities that, while extremely difficult to be detected via traditional testing, provide the means for attackers to compromise the application and potentially take over the host. Moreover, applications developed using vastly different development methodologies and quality often coexist in the same environment (e.g. entertainment systems in cars coexisting with critical systems such as stability control, steering and breaking) allowing stepping stone attacks.

The need to protect these applications from being contacted by random hosts from across the Internet has been well understood since the original Morris worm of the mid-80s. Unfortunately, the solutions proposed proved inadequate and in some cases exacerbated the problem. For example firewalls have the ability to guard the entrances of networks and allow only "good" guys to connect to internal hosts. Initially, firewalls proved quite effective, but over the years, they gradually became irrelevant (Niederberger et al., 2006) as WiFi connections allowed firewalls to be bypassed and Transport Layer Security (TLS) (Dierks and Rescorla, 2008) ensured that traffic flowing via these firewalls is encrypted so that they are incapable of monitoring it for attacks.

In a typical banking application scenario, the customer uses TLS to connect to the banking application running on the banks servers (Hiltgen et al., 2006). The connection is cleared by the firewall before the customer is identified by the system. This is so because identification is handled by the application after the TLS session has been established. Thus, not only we are forcing the firewall to allow all connections from the public Internet to our application, but we also prevent any network-based intrusion detection system from monitoring the data exchanged over the encrypted connection.

It is thus crucial for the connection request to be vetted before it is allowed to reach the application. The problem with the traditional way of controlling access to the application, is that it involves a lot of

https://doi.org/10.24355/dbbs.084-201806251531-0

manual configurations (e.g. configuring packet filters etc.), requires administrator access and is, thus, expensive, slow and error prone. We, therefore, need a faster, automated way for access control. We propose to de-couple the access control mechanism from the network code and use a policy engine to evaluate requests if authorized, and to reconfigure the system accordingly. In this way we can accommodate dynamic scenarios such as load balancing or failure recovery.

Our mechanism should also allow interactions between applications running in distributed control systems such as those found in vehicles, airliners, etc. Numerous hacking demonstrations on car ECUs have shown that once access to the internal communication bus is gained by the attacker, then the entire vehicle is compromised (Koscher et al., 2010). Traditional approaches have proven to be too inflexible for complex distributed environments so we had to look for a better solution based on the distributed firewall model proposed by Ioannidis et al (Ioannidis et al., 2000). Under this framework each connection request must include appropriate authentication and authorization to allow both the sending and receiving hosts to decide whether to allow the connection request to go through. In this way, there is no need to pre-configure the elements of the distributed system with access control information, but the communications policy is dynamically constructed as communications requests are made and granted. Eventually we will have a set of secure links connecting applications running on different hosts, but unlike the static configuration model, new requests can be accommodated and integrated into the overall system policy.

Any distributed security paradigm will eventually fail unless it can be used by the application developers. By providing a proper security application-layer developers will be able to implement applications that use secure communications efficiently. Moreover, this layer will make it more convenient and applicable for protecting the application communications.

In this paper, we present new APIs that give any application the power to control its security policy by providing sufficient configurations to the security layer. This enables an application to get the relevant information about the applied security mechanisms and all the parameters of the secure channel. We also provide APIs which allow any application to authenticate the requesters identity and indicate whether this request is authorized or not based on the security policy of the receiving application. The authorization decision will not be based on packet filtering and Access Control Lists (ACLs) mechanisms. We use the Keynote trust management (Blaze et al., 1999) model to enable the application to determine the allowed network access regarding credentials presented by the remote application, which should conform to the local policy of the application.

The rest of the paper is organized as follow. In Section 2, we provide a short background and some related works. Section 3 explains the design of the secure APIs. The implementation of APIs is detailed in Section 4. In Section 5, we discuss some evaluation asspects. Finally, Section 6, contains some concluding remarks.

2 BACKGROUND AND RELATED WORK

Many mechanisms and protocols were defined to protect sensitive and critical system resources. TLS is one of these techniques which is used to secure communications at the application layer. It uses encryption and authentication to keep communications private between two devices; typically a web server (website) and a browser. TLS is limited to TCP and STCP based protocols. An adaptation of TLS for UDP protocol is available; it is called DTLS. However, it is not widely used. Numerous modifications on the application's source code are required to run it over TLS. In some circumstances, these changes impose significant complexity overhead.

Another important mechanism is Internet Protocol Security (IPsec) (Kent and Seo, 2005). It was designed to provide network security services to protect Internet datagrams. It provides its security services over two protocols: the first one is called Authentication Header (AH) (Kent and Atkinson, 1998a) which provides origin authentication, data integrity and optional replay attack protection. The second protocol is called Encapsulating Security Payload (ESP) (Kent and Atkinson, 1998b) which provides the confidentiality and authentication of the exchanged data. Although IPsec was introduced a few years ago, its usage was confined to VPN implementation. The lack of APIs was one of the main reasons that limited the adoption of the IPsec to provide end-to-end security (Bellovin, 2009), (Ioannidis, 2003). Without these APIs, applications were not able to interact with the IPsec layer and verify whether IPsec services are being used underneath. The requirements of an application to interact with security layer were specified by Richardson and et al. (Richardson and Sommerfeld, 2006). They claimed that each application should be able to:

1. Determine HOW a communication was protected,
2. Identify WHO is the remote party,

https://doi.org/10.24355/dbbs.084-201806251531-0

3. Influence HOW the protection should take place, and
4. Indicate WHY an authorized communication failed.

All these requirements should be carried out as a set of APIs which could be used by the applications.

McDonald (McDonald, 1997) first proposed the implementation of such APIs. He introduced the concept of IPsec API as an extension to the BSD Sockets where applications could provide their configuration per socket. Wu et al. (Wu et al., 2001) provided an API which gives the application the capability to choose the IPsec tunnel which will be used to protect outgoing packets. It also enables the application to know which tunnel was used to receive a particular incoming packet. Information are extracted from the IPsec header, associated with the received data and then delivered to the application regarding the used protocol (TCP, UDP). However, their APIs did not manage the creation of the IPsec tunnels by the application.

According to Arkko et al. (Arkko and Nikander, 2003), the lack of APIs is not the only problem in the IPsec design. The current security policy mechanism is another stumbling block the adoption of IPsec faces, in the end to end application-level protection.

Yin and Wang in (Yin and Wang, 2007) suggested a solution fix for the lack of the IPsec policy system by introducing a mechanism that makes applications aware of the IPsec policy. They implemented a socket manager which detects the sockets' activities of the running application and report them to the application policy engine. The engine gets the application policies, which are stored in the policy repository. It processes them and then writes fine-grained policies into the Security Policy Database (SPD). In their work, they did not change the existing IPsec/IKE infrastructure. They kept using host authentication rather than application-based authentication which is required to secure application communications. Pereira et al. (Pereira and Beaulieu, 1999) tried to provide a user-based authentication schem within the IKE implementation by introducing new a exchange phase, which was called p̈hase 1.5.̈ Within this invented phase, challenge-response messages are exchanged between the remote user and the security gateway.

3 DESIGN

Although IPsec's application-agnostic design is an advantage, in the sense that IPsec can protect applications without any modification to their source code, it is also a disadvantage because (a) the application can not request a secure connection, (b) it can not specify parameters of the secure connection, (c) and it can not verify whether its connection is secure or not. In this work, we try to handle some of the obstacles by providing suitable APIs which enable the application to so.

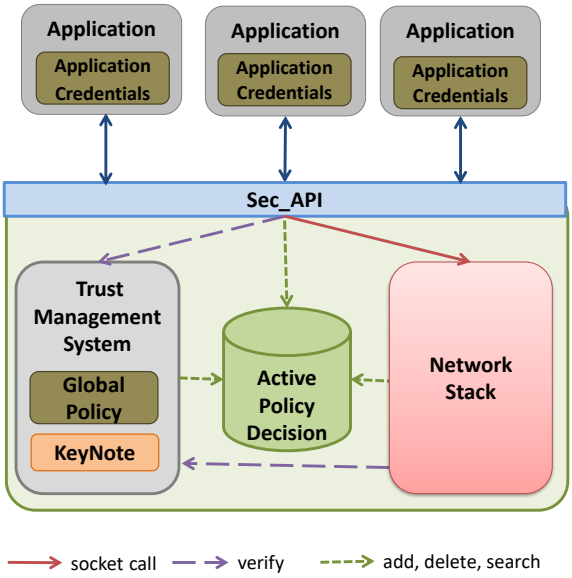

Figure 1: Application interactions using the secure API to interact with the trust management system and the security layer in the network stack (i.e. IPsec)

```
Authorizer:    Trusted_Public_key
Licensees:     App_Public_key
Conditions:  ( Src_ip_address  ==ANY
             && Dst_device_name == My IP
             &&  Src_port  == ANY
             &&  Dst_port  ==  ( 80 || SSL PORT)
             && Security_level  >= SL_INTEGRITY  && Priority_level  ==  HIGH
              -> "ALLOW"
Signature:     Trusted_Private_key
```

Figure 2: An example of a security credential which was given to an application

3.1 Overriding The Security Policy

Using static security policies is one of the main drawbacks of the existing security implementation. For example, SPD selectors which are used with IPsec should be specified in advance. These selectors contain the IP addresses and port numbers. In some circumstances, some selectors are not available in advance, such as the port number is not always static for some protocols. Consequently, such selectors are left empty in the IPsec security policy. Later, when two different applications on the same host with different port numbers, try to connect to the same remote application, there will be no distinguishable selectors to

https://doi.org/10.24355/dbbs.084-201806251531-0

differentiate their two separate connections. Thus, the need of dynamic management for the IPsec policy is critical.

Supplying the application with the ability to define the security policies might solve such a problem. Each application should be authorized to set up and update its security policy which fulfills its requirements dynamically. Authorization is required to prevent the application from tampering with the security policy parameters which belong to different applications. The decision whether the application admitted setting the security policy is made by the trust management system. The trust management system, shown in Figure 1, contains KeyNote engine which evaluates the potential action of each application. The request should be consistent with the global system policy, to be authorized. The global system policy represents the least accepted security parameters and system-wide security configuration. Each application has communication credentials, as shown in Figure 2, which assure the ability to initiate an authorized connection as long as it does not conflict with the global policy. Dynamic management will ensure interdependence between the policy rule and the connection's characteristics. To deal with the dynamic port number problem, we bind each application's connection to a port from the static ports pool. In the same way, if the IP address is left empty in the application policy, it will be assigned to the host IP address. When an application would like to set up a connection with a remote one by *Sec_bind*, *Sec_connect*, *Sec_sendto*, or *Sec_recvfrom* it will provide its security context which contains attributes of the proposed action as shown in Table 1.

All the proposed functions are using the same parameters of the comparable Socket functions, besides their credentials and the security context:

```
int Sec_connect(connect() parameters,
char* credential,security_context* sec_cont)
int Sec_bind(bind() parameters,
char* credential,security_context* sec_cont)
int Sec_sendto(sendto() parameters,
char* credential,security_context* sec_cont)
int Sec_recvfrom(recvfrom() parameters,
char* credential,security_context* sec_cont)
```

Security context and the credential (it could be chain of credentials) are, all together, delivered to the trust management system which will check whether an application is authorized to set up such a connection or not. It will determine whether the proposed action is compliant with the global policy by applying the security context parameter to the credential's conditions. In the case of an authorized request, a new security rule will be added to the SPD. In addition to that, a reference from this rule will be stored in the protocol control blocks (PCBs) structure associated with the socket. This reference will be used later to delete this rule when the application closes the connections (i.e. *Sec_close*).

```
int Sec_close{socket s )
```

3.2 Interacting With Security Layer

By placing the IPsec mechanisms in the network layer, the designer created an abstraction layer that decouples the applications from the security of its communications. In other words, the applications cannot easily determine the identity of their communicating parties or the characteristics of the security selected on the communication link. By contrast, if the applications use TLS, it can both authenticate their remote party and specify the form of encryption used for the communication.

Providing sufficient APIs, which enable the application to interact with the IPsec layer, will increase the application comprehension about the used security mechanisms in the ground layers. The proposed APIs enable any application to determine if an incoming communication is protected or not. In the case of protection, it grants an application the right to retrieve the security properties which were used to protect the data. These security properties could contain the cryptographic algorithms which were used, the length of the key, the type of security service (confidentiality, or integrity) etc. Each application will be authorized to get the security parameters which relate to its connection. The default security settings, which is defined in the global policy, are available to any application.

The *Sec_getsockopt* and *Sec_setsockopt* system calls manipulate the options associated with a socket. Table 2 describes the security services which could be used : Each of the options presented in Table 2 can have different security levels associated with it (i.e. SEC_NONE, SEC_BYBASS, SEC_USE, etc.). The relevant information (security levels, security service, etc.) is stored in PCB's structure associated with the socket.

```
int Sec_getsockopt(socket s, Sec_service srv,
Sec_level lvl, void *optval,int *optlen)
int Sec_setsockopt(socket s, Sec_service srv,
Sec_level lvl, void *optval,int *optlen))
```

3.3 Providing The Ability Of Authorization and Authentication

The existing certificate authority does not assure the trustworthiness of the key owner, but merely authenticates the owner's identity. However, we are not only interested in identifying the remote part. We need a

https://doi.org/10.24355/dbbs.084-201806251531-0

Table 1: Security Context Fields.

Field	**Description**
Ports numbers	Port numbers which application listens to and the remote port number (it can be any)
IP_addresses	IP address used by the application and the remote IP (it can be any)
Issued_Time	Time when the request was issued, it is used to prevent replay attacks
Others	Another security properties such as encryption algorithm, level of security, etc.
Signature	Requester application signs all the previous fields with its private key

Table 2: Security Services.

SEC_AUTH	Refers to the use of authentication for the connection
SEC_CON	Refers to the use of encryption for the connection
SEC_AUTH_CON	Refers to the utilization of both authentication and encryption

mechanism that tells us the allowed actions that the remote part is able to perform . The certificate infrastructure does not handle the decisions whether the remote party is allowed to access services or not. This decision is left up to the application.

Within the proposed APIs layer, we give the application the ability to authenticate and decide whether the remote peer is authorized to do some actions depending on its credentials and the local policy in an easy manner. In other words, we combine the authentication with access control. The application is able to get the remote party's identity and credentials.

```
int Sec_getremoteCredentials(socket s,
        char * credentials)
int Sec_getremoteKey(socket s, char * key)
```

4 IMPLEMENTATION

We started our implementation by providing a set of APIs which function on the socket layer of the network stack and interact with the IPsec layer. Practically, in the monolithic operating systems (OS), such as Linux and Windows, IPsec and the network stack are implemented in the kernel; this limits our ability to develop our APIs. On the other hand, Within Microkernel OS, IPsec, which is integrated into the TCP/IP stack, is implemented in the user space. This gave use more scalability and functionality to implement the required APIs. Thus, we implemented our APIs on a Raspberry PI computer running a Microkernel OS; we used Genode OS (Genode Labs GmbH,).

We used an existing embedded IPsec infrastructure, which was implemented on Genode OS by Hamad et.al (Hamad and Prevelakis, 2015), to develope our security APIs. For the policy framework, we used a framework that was proposed by Prevelakis et.al (Prevelakis and Hamad, 2015) to provide a mechanism to maintain the integrity of the applications credentials and enforce them during the runtime phase. Keynote library was ported to Genode OS which was used to evaluate the applications requests. We implemented many APIs to provide the intraction between the trust managment system and the active policy repository (as shown in hown in Figure 1).

5 EVALUATION

Security has a price; it is clear that using our APIs imposes some overhead. Such overhead comes from policy evaluation, credential verification and providing security services (i.e. IPsec). From the various security overheads, we have found in (Hamad and Prevelakis, 2015) that the operational security overhead of security services is very low after the configuration is over.

In the other hand, the overhead of evaluating the credentials and the operation of the policy engine is significant, but it occurs only when a new connection request must be considered (i.e. it is compliant with the policy). Hence, the cost must be amortized over the lifetime of the connection. In vehicular platforms, connections tend to have a long lifetime. So, this particular overhead is not significant. Moreover, various optimization techniques, such as caching verified credentials and policy, may be used to reduce the cost of policy evaluation.

6 CONCLUSIONS

Providing the proper security application-layer can increase the ability of applications to use secure communication efficiently. In the same time, it will make it more convenient and applicable for protecting the application communications.

In this paper, we implement new APIs which give any application the ability to control its security pol-

https://doi.org/10.24355/dbbs.084-201806251531-0

icy by providing sufficient configurations to the security layer. Moreover, it enables an application to get the relevant information about the applied security mechanisms and all the parameters of the secure channel. We also provide APIs which allow any application to authenticate the requester's identity and indicate whether this request is authorized or not, based on the security policy of the receiving application. The authorization decisions are not based on packet-filter and ACLs mechanisms.

Adoption of our secure APIs by applications caused a performance overhead. However, our implementation is still in a proof-of-concept stage; different optimization methods could be used to reduce this overhead.

7 ACKNOWLEDGEMENT

This work was supported by the DFG Research Unit Controlling Concurrent Change (CCC), funding number FOR 1800. We thank the members of CCC for their support

REFERENCES

Arkko, J. and Nikander, P. (2003). Limitations of ipsec policy mechanisms. In *Security Protocols, 11th International Workshop, Cambridge, UK, April 2-4, 2003, Revised Selected Papers*, pages 241–251.

Bellovin, S. (2009). Guidelines for specifying the use of ipsec version 2. BCP 146, RFC Editor.

Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A. D. (1999). The keynote trust-management system version 2. RFC 2704, RFC Editor.

Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard).

Genode Labs GmbH. *Genode OS Framework*. `https://genode.org/` [last access on Jan 2017].

Hamad, M. and Prevelakis, V. (2015). Implementation and performance evaluation of embedded ipsec in microkernel os. In *Computer Networks and Information Security (WSCNIS), 2015 World Symposium on*, pages 1–7. IEEE.

Hiltgen, A., Kramp, T., and Weigold, T. (2006). Secure internet banking authentication. *IEEE Security & Privacy*, 4(2):21–29.

Ioannidis, J. (2003). Why don't we still have ipsec, dammit? In *NDSS 2003*.

Ioannidis, S., Keromytis, A. D., Bellovin, S. M., and Smith, J. M. (2000). Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199. ACM.

Kent, S. and Atkinson, R. (1998a). Ip authentication header. RFC 2402, RFC Editor.

Kent, S. and Atkinson, R. (1998b). Ip encapsulating security payload (esp). RFC 2406, RFC Editor.

Kent, S. and Seo, K. (2005). Security Architecture for the Internet Protocol.

Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al. (2010). Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE.

McDonald, D. L. (1997). A Simple IP Security API Extension to BSD Sockets. Internet-Draft draft-mcdonald-simple-ipsec-api-02, Internet Engineering Task Force.

Niederberger, R., Allcock, W., Gommans, L., Grünter, E., Metsch, T., Monga, I., Valpato, G. L., and Grimm, C. (2006). Firewall issues overview.

Pereira, R. and Beaulieu, S. (1999). Extended Authentication Within ISAKMP/Oakley (XAUTH). Internet-Draft draft-ietf-ipsec-isakmp-xauth-06, Internet Engineering Task Force. Work in Progress.

Prevelakis, V. and Hamad, M. (2015). A policy-based communications architecture for vehicles. In *International Conference on Information Systems Security and Privacy, France*.

Richardson, M. and Sommerfeld, B. E. (2006). Requirements for an IPsec API. Internet-Draft draft-ietf-btns-ipsec-apireq-00, Internet Engineering Task Force.

Wu, C.-L., Wu, S. F., and Narayan, R. (2001). Ipsec/phil (packet header information list): design, implementation, and evaluation. In Li, J. J., Luijten, R. P., and Park, E. K., editors, *ICCCN*, pages 206–211. IEEE.

Yin, H. and Wang, H. (2007). Building an application-aware ipsec policy system. *IEEE/ACM Transactions on Networking*, 15(6):1502–1513.

https://doi.org/10.24355/dbbs.084-201806251531-0